EPC
Applied PCs

Fred Eady

Rabbit Season

Part 5: A Board You Can Afford

In true rabbit fashion, this project has resulted in quite a proliferation of articles. But Fred's got one last hop for the Rabbit dev board project. The jump to Internet connectivity was as simple as a twitch of the nose.

You know, having that Rabbit and Alice around has really made things interesting lately. And, another old friend, Grace Slick of Jefferson Starship, took some time last week to stop by the Florida room to say hello to Alice, the inspiration for her song, "White Rabbit," whom she hasn't seen in some 30 years. It's hard to work with all the music in the air.

Anyway, when thinking about this article, I came to a conclusion. Now that I've got a couple of *Circuit Cellar Online* articles behind me, it's obvious that I'm part of a powerful technical information outlet. I'm sure that if the *Circuit Cellar* columnists you follow every month in the print version were given unlimited article space, you'd probably be enjoying a magazine as thick as a phone book. Fortunately, the web site allows me and other authors to provide twice as much information each month.

After receiving some beta PPP networking code from Rabbit Semiconductor's network guru Charlie Krauter, it looks like the PPP hardware and all of the construction details I promised last month will appear online.

LET'S GO TO THE HOP

This is the last installment in the Rabbit series and, just as I moved the Seiko PPP IC and ISOModem to the web, I am moving the Rabbit 2000 Core Module to print. Everything you need to make the Rabbit 2000 IC the star of your design is included on the Rabbit 2000 Core Module, and is designed to be plugged in. This eliminates having to design and lay out the microcontroller engine part of your project. Because it's my job to make your job easier, Photo 1 is a shot of the Rabbit 2000 Core Module holding court on its development board.

Photo 2 shows that, in addition to the Rabbit 2000 Core Module, I've assembled a little plug-in module of my own. It consists of two parts, a MAXIM 235 and a 1-µF tantalum capacitor. The MAX235 is an RS-232-to-TTL interface IC used to convert modem signal levels to TTL coming into the Rabbit 2000 Core Module and TTL-to-RS-232 signal levels going out. It's a standard implementation, as is seen in Figure 1. Because the name of the game is dialup and it doesn't have

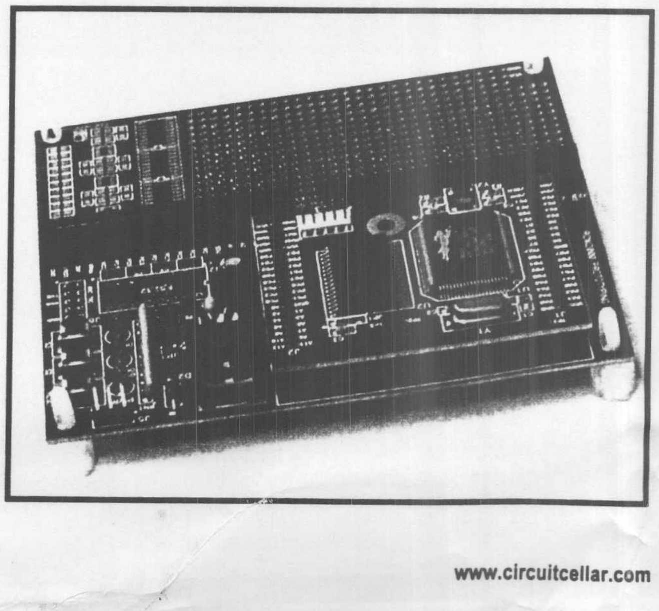

Photo 1—*I like this setup a little better than the Jackrabbit. The Core Module allows more flexibility as to what you can directly connect to it.*

to be fast, I pulled an old U.S. Robotics 28.8 external modem off the shelf for this project.

ONE HOP TO THE INTERNET

To check out *Circuit Cellar Online* if you aren't using a cable modem or DSL, dial a phone number to connect to your ISP and open the window to the 'Net. If you use a dial-up modem for Internet access, most likely the protocol you use to establish contact with the ISP is PPP, or point-to-point protocol. I'll use the same PPP rules that your desktop computer uses to get a phone to the Rabbit's ears and a call through to the Internet.

The mission this time is to get the Rabbit on the phone, and then on the Internet. After that, you can instruct it to send e-mail or serve HTML pages. The problem is, there are no predefined Dynamic C PPP libraries or routines, but this setback was quickly eliminated with a single e-mail to Rabbit Semiconductor. It seems that Charlie was working on the PPP code as I was working on this article.

PPP is taken for granted. But, PPP is an important step in the Internet access process. Like everything else in life, you don't appreciate it until you can't get it. With that, I'm going to delve into what makes PPP tick and some of the code necessary to hop around in the Internet.

PPP

A dial-up session to your ISP is defined as a point-to-point link (i.e., from your modem to a modem at the ISP location). When the physical modem-to-modem session is established, you can't just pass packets to the interface in any way you choose. There has to be some standard of communications, an agreement between the sending and receiving parties. This normally involves the determination of which protocol will be used and the maximum size of the packets that will flow between the peers within the constraints of the selected protocol.

The point-to-point protocol was designed to provide a standard method for transporting multi-protocol datagrams over simple point-to-point links. A simple point-to-point link is usually a full duplex connection that delivers packets in the order that they are transmitted. A datagram is the unit of transmission in the network layer (e.g., IP) and may be encapsulated in one or more packets before being passed down to the data link layer.

PPP was designed for easy use and implementation. To make the concept of PPP clearer, you can break it down into three main components—encapsulation, a link control protocol (LCP), and a collection of network control protocols (NCPs).

A packet is the basic unit of encapsulation. Packets are passed between the network layer and the data link layer. Because the data link layer is responsible for creating frames, a single packet or multiple packets are usually the cargo of frames. PPP uses a frame format and protocol similar to the HDLC (high-level data link control) frame, which begins and ends with a flag or sync character 0x7E. The combination of encapsulation coupled with framing allows different protocols to be transported over a PPP link.

Field length in bytes	1	1	1	2	Variable	2 or 4
	Flag	Address	Control	Protocol	Data	FCS

Figure 2—*You just visited a similar "field" in the world of Ethernet.*

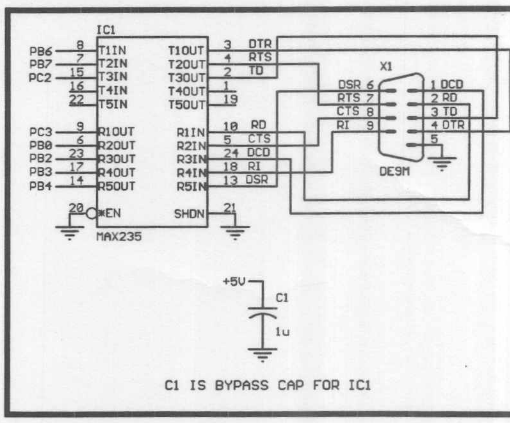

Figure 1—*There's nothing fancy or unknown here. This circuit has been used for years.*

RABBIT IN A FIELD

Figure 2 shows that a standard PPP frame consists of six fields. Let's examine them left to right. I've already mentioned the flag field, a single byte that sits at the beginning and end of a frame. If you're wondering what would happen if your data contained a 0x7E, not to worry. PPP has the capability to escape such characters. In fact, PPP escapes characters less than 0x20. This escape process involves exclusive ORing the control code with 0x20. For instance, to send 0x7E as data, 0x7E would be exclusive ORed with 0x20, yielding 0x5E. Then the escape character 0x7D is inserted before the 0x5E, resulting in a two-byte sequence of 0x7D and 0x5E, representing the data value of 0x7E. At this point, the receiving machine simply does another exclusive or with 0x20 against the 0x5E and discards the 0x7D escape character to return the original value of 0x7E.

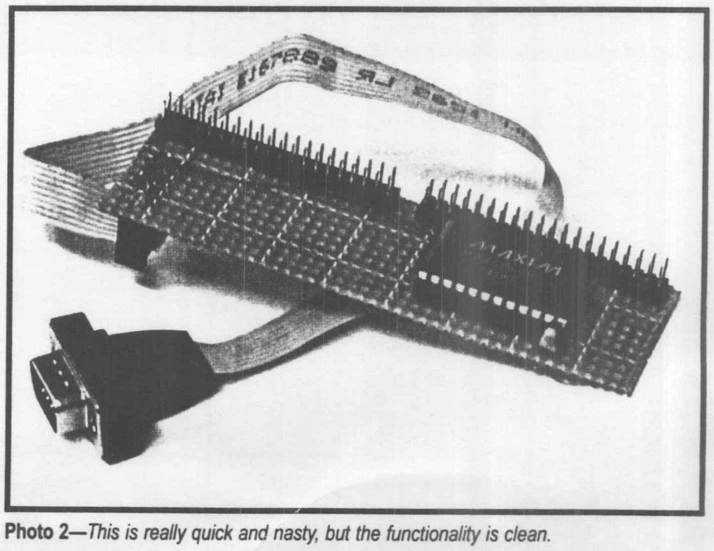

Photo 2—*This is really quick and nasty, but the functionality is clean.*

PPP has no method for assigning individual station addresses. So, the Address field is a standard broadcast address of 0xFF. PPP is connectionless (i.e., it doesn't require a formal session with sequenced frames). The 0x03 in the Control field specifies this as it represents the transmission of data using an unsequenced frame. If you were to dig out the bit scheme, you'd find that the 0x03 is the unnumbered information (UI) command with the poll/final (P/F) bit set to zero.

Knowing what's in the information field of a PPP frame is important. The two bytes of the protocol field identify the protocol that's encapsulated in the information field. Table 1 states some of what you can put into the protocol field and what the values represent with the protocol. All values are in hexadecimal.

0001 to 001f	Reserved (transparency inefficient)
0021	Internet protocol
0023	OSI Network layer
0025	Xerox NS IDP
0027	DECnet phase IV
0029	Appletalk
002b	Novell IPX
002d	Van Jacobson compressed TCP/IP
002f	Van Jacobson uncompressed TCP/IP
0031	Bridging PDU
0033	Stream protocol (ST-II)
0035	Banyan vines
0037	Reserved (until 1993)
00ff	Reserved (compression inefficient)
0201	802.1d Hello packets
0231	Luxcom
0233	Sigma network systems
8021	Internet control protocol
8023	OSI Network layer control protocol
8025	Xerox NS IDP control protocol
8027	DECnet Phase IV control protocol
8029	Appletalk control protocol
802b	Novell IPX control protocol
802d	Reserved
802f	Reserved
8031	Bridging NCP
8033	Stream control protocol
8035	Banyan vines control protocol
C021	Link control protocol
C023	Password authentication protocol
C025	Link quality report
C223	Challenge handshake authentication protocol

Table 1—*During the debugging process, I got to know the C021 code well.*

The Data, or information field, is interesting because it can be zero bytes in length. The default maximum length of this field is 1500 bytes (this is negotiable). The Data field can also be padded. An easy way to find the end of the Data field is to find the flag at the end of the frame and count back past the two FCS bytes in the Frame Check Sequence field.

Usually, where you find fields that comprise a frame, you find a protocol that goes with them. LCP is the PPP method of establishing, configuring, maintaining, and terminating the point-to-point connection. LCP packets are akin to marines because they are the first to fight. Each peer participating in a PPP link sends LCP packets to configure and test the data link before establishing communications over the PPP link. You can be authenticated only after the link is deemed satisfactory. So, authentication comprises supplying the log-on user ID and password to the remote peer.

Code	Identifier	Length	Data
1 byte	1 byte	2 bytes	Variable

Table 2—*This may look like any other field I'm discussing with you today, but remember that encapsulation is the key to making it all work.*

After the physical link is made, PPP then sends NCP packets to choose and configure one or more network-layer protocols. After each desired network-layer protocol has been configured, datagrams from these protocols can be exchanged between the peers on the link. The link configuration will exist until LCP or NCP packets close it down or an operator or predefined timer stops the link.

PHASES

To better understand LCP, you can divide its functionality into four phases. As I stated earlier, the first operation is physical link establishment. In Figure 3, link dead is the starting and ending point of the phase relationships. The modem is commanded to dial and subsequently contacts the answering modem. Carrier detect from the modem on your end signals the LCP that the physical link is active and ready to be used.

The UP signal from the physical layer puts the LCP process in the link establishment phase. Before any datagrams can be exchanged over the newly opened link, LCP opens the connection and negotiates configuration parameters. This phase is complete when a configuration-acknowledgment frame (Configure-Ack) has been both sent and received. All of the configuration options are assumed as defaults until altered by the negotiation process. These configuration options are not protocol-dependent. The NCP handles any protocol-dependent configuration issues in the network-layer protocol phase.

If necessary, you are authenticated before any network-layer protocol datagrams are exchanged between the peers on the point-to-point link. PPP supports two authentication protocols. Password authentication protocol (PAP), a simple method of establishing user identity, is the most popular. Following the establishment of the PPP link, the user ID and password are sent to the remote peer until the authentication is positively acknowledged or the link is broken. With PAP, everything is sent in the clear with no encryption. This leaves the remote peer in charge of determining how to affect some sort of log-on security. The second authentication protocol is challenge handshake authentication protocol (CHAP). This method sends a

1	Configure-Request
2	Configure-Ack
3	Configure-Nak
4	Configure-Reject
5	Terminate-Request
6	Terminate-Ack
7	Code-Reject
8	Protocol-Reject
9	Echo-Request
10	Echo-Reply
11	Discard-Request
12	RESERVED

Table 3—*These are bounced back and forth between peers many times during a session.*

challenge message to the remote peer, which responds with a calculated value. If the sender agrees with the returned value, the authentication is positively acknowledged. Because this calculated value is always unique, CHAP obviously offers a higher level of log-on security than PAP. That's all good, but by default, authentication is not necessary.

After the link is opened and the user or peer is authenticated, network-layer protocol configuration negotiation occurs. Each desired network-layer protocol (IP, IPX, etc.) is configured separately by the appropriate NCP. As long as the PPP link is up to this point, each individual protocol can be brought up and taken down at any time by its controlling NCP. If LCP closes the link, the network-layer protocols are alerted so that they can take appropriate action.

LCP can terminate the link at any time. Normally, this is planned and executed by you or the administrator. Other physical events, such as the expiration of an idle-period timer can also terminate the link.

Each LCP phase employs a particular frame. Link-establishment frames consisting of link configuration packets (Configure-Request, Configure-Ack, Configure-Nak and Configure-Reject) are used to establish and configure a link. Link-termination frames containing link termination packets (Terminate-Request and Terminate-Ack) are used to terminate a link, and link-maintenance frames filled with link maintenance packets (Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply, and Discard-Request) are used to manage and debug a link.

Table 2 is the format of an LCP packet. The Code field denotes the type of LCP packet. For instance, 0x01 in the Code field shows that the LCP packet is a Configure-Request packet. A list of the Code field entries is shown in Table 3.

The Identifier field is used to match requests and replies to the peers. If this field is invalid, the packet is ignored without any action springing from its contents.

The Length field includes the Code, Identifier, Length, and Data field byte counts. Everything outside the limits defined by the Length field entry is treated as padding and thrown away.

The Data field contents are directly dependent on the Length field byte count. Also, the format of the Data field is controlled by the type of packet defined in the Code field. Typically, the Data field format follows the Type/Length/Data layout. A list of LCP option type field entries is shown in Table 4.

As all of these various types of frames are flying among peers, commands to configure and send the aforementioned packets and frames are issued depending on the conditions at a particular point in the execution of a phase. These packet-assemble and packet-send commands are generated by events, actions, and state transitions. This collection of events, actions, and state transitions (as they relate to PPP) are known as the option negotiation automaton.

Without going into lengthy discussion, the option negotiation automaton is a definition and model of how the states and state transitions must be made to follow the rules in implementing PPP. PPP code is written using the option negotiation automa-

Listing 1—*This is a dump snippet. The idea is to convey the fields and how they all play together with the help of encapsulation.*

```
AT
OK
ATZ
OK
Modem  Initialized

ATDT6334710
CONNECT  9600/ARQ/V32/LAPM/V42BI

**  Florida  Online  MAX-ROC9  **


login:edtp
password:
      Entering  PPP  Mode.
      IP  address  is  208.14.41.96
     MTU  is  1524.

Sent:  FF  03  C0  21   01  08   00  18   01   04   05  DC   02   06   00
 00   00   00   05   06
Got  LCP  Request
        FF  03  C0  21   01  01   00  3C   01   04   05  F4   02   06   00
 0A   00   00   07   02

Sent:  FF  03  C0  21   04  01   00  0E   01   04   05  F4
Sent  LCP  Reject
Received  LCP  Config-ACK
Got  LCP  Request
Sent:  FF  03  C0  21   02  02   00  12   01   04   05  F4   02   06   00
 0A   00   00   07   02
LCP  Config  Acked

PPP  established

IP  address  is  208.14.41.96


Sent:  FF   03  C0  21  05   0A   00   04   00  FE   06  C3  51  D0   0E  29
 60   00   00   00
Terminate  Ack

NO  CARRIER
ATH
OK
ATZ
OK
```

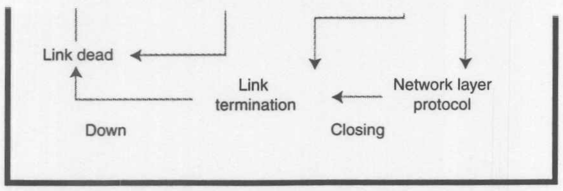

Figure 3—*I'm glad it doesn't use "dead" for low and "alive" for high in the logical sense.*

ton as the flowchart. Now that you're dangerous, let's look at a piece of PPP protocol and break down the fields to determine what went right and what went wrong in Listing 1.

BREAKING IT DOWN

Listing 2 is a code snippet that runs to produce the AT and ATZ modem commands. In terms of phase, you are at dead here. The modem initializes and the ISP is dialed. When the carrier detect is sensed, the establish phase is entered. In the meantime, the ISP asks for a login and password. This is not a PPP function. The Rabbit 2000 Core Module is assigned an IP address and given the MTU size.

The next line is where the real PPP fun begins. Let's break down the trace. The 0x7E is really there on the datascope, but the debug code in the Rabbit library doesn't display it. So far, everything is just as it should be. The flag byte is followed by 0xFF in the Address field and 0x03 represents un-sequenced frame transmission in the Control field.

The next two octets (there's Mozart again) define the protocol encapsulated in the packet. 0xC021 is the assigned number for LCP. The following fields are Data fields formatted according to the protocol called out in the protocol field (0xC021). Another look at Table 2 shows that, for an LCP packet, you should expect a single-byte Code field. This Code field is populated with 0x01. This is a Configure-Request packet. The next byte is the Identifier, which, in this case, is 0x08 followed by the two-octet Length field containing 0x0018. This trace doesn't show all of the bytes between flags, but I verified on the datascope that there in from the end flag).

The next byte is the LCP Configuration Options Type field. The 0x01 shows that this is an MRU setting. The length of this MRU segment is 0x04 octets and the value of the MRU (maximum receive unit) is 0x05DC, or 1500 decimal. Again, following the Type/Length/Data layout and beginning with the next byte, 0x02, the async control character map is being negotiated and you're asking for no escaped characters with the four octets of zeroes. The 0x05 and 0x06 bytes that follow are the beginning of sending what is called the magic number. The magic number is an arbitrary number derived from any means that is as random as possible. The idea is to

5	Magic-Number
6	Reserved
7	Protocol-Field-Compression
8	Address-and-Control-Field-Compression

Table 4—*The magic number is akin to a boomerang in that it is thrown out to see if it returns.*

Listing 2—*I'm really showing you this so you can hook up your Rabbit 2000 Core Module to a modem.*

```
/*******************
external_modem.lib

A set of routines for controlling an external modem through a
 full RS232 link.
Default hardware is a core module using serial port C
Pin assignments for control lines are:
        DTR - PB6 (out)
        RTS - PB7 (out)
        CTS - PB0 (in)
        DCD - PB2 (in)
        RI  - PB3 (in)
        DSR - PB4 (in)
        TD  - PC2 (out)
        RD  - PC3 (in)

*****************************/


/*** BeginHeader */
//#ifndef __DCSPPP_LIB
//#define __DCSPPP_LIB
#ifndef __EXTMODEM_LIB
#define __EXTMODEM_LIB

unsigned long extmodem_baudrate;
//current baud rate for communication with modem

//setup flow control for port C
#define SERC_RTS_PORT PBDR
#define SERC_RTS_SHADOW PBDRShadow
#define SERC_RTS_BIT 7
#define SERC_CTS_PORT PBDR
#define SERC_CTS_BIT 0

/*** EndHeader */

/*** BeginHeader ModemOpen */
int ModemOpen(unsigned long baud);
/*** EndHeader */

/* START FUNCTION DESCRIPTION
*********************************************
```

(continued)

Listing 2–*continued*

```
ModemOpen
 <EXTERNAL_MODEM.LIB>

SYNTAX:                        int  ModemOpen(unsigned  long
 baud);

DESCRIPTION:  Starts  up  communication  with  an  external  modem

PARAMETER1:          The  baud  rate  for  communicating  with  the
 modem

RETURN  VALUE: 1  -  External  modem  detected
               0  -  not  connected  to  external  modem
END  DESCRIPTION
 ***********************************************************/

int  ModemOpen(unsigned  long  baud)
{
        extmodem_baudrate  =  baud;
        //check  DSR  line
        if(ModemReady())
        {
                serCopen(baud);
                serCflowcontrolOn();
                return  1;
        }
        else
        {
                return  0;
        }
}

/***  BeginHeader  ModemInit  */
int  ModemInit();
/***  EndHeader  */

/*  START  FUNCTION  DESCRIPTION
 ******************************************
ModemInit
 <EXTERNAL_MODEM.LIB>

SYNTAX:                        int  ModemInit();

DESCRIPTION:  resets  modem  with  AT,  ATZ  commands

RETURN  VALUE: 1  -  success
                                      0  -  modem  not  responding
END  DESCRIPTION
 ***********************************************************/

int  ModemInit()
{
        int  i;

        for(i  =  0;i  <  3;i++)
        {
                ModemSend("AT\r");
                if(ModemExpect("OK",  2000))
                {
                        ModemSend("ATZ\r");
                        if(ModemExpect("OK",  2000))
                        {
                                return  1;
                        }
                }
        }
        return  0;
}
```

send this and see what you get back. If your magic number is returned to you, then you are in a loop-back mode. On the other hand, if you get a totally different number than what you concocted and sent out, that probably means the link is ready for use. If the magic number is not negotiated, it consists of four octets of zeroes.

The next line after "Got LCP Request" is incoming from the peer at my ISP. Notice the differences in the MRU and async control character map entries. The Rabbit rejects the Configure-Request with a 0x04 (Configure-Reject) in the Code field. The ISP peer then says OK and sends a Configure-Ack with settings you can agree on. At this point, you should be OK with the negotiated values on your end, so you send a positive acknowledgement to the ISP peer. The PPP link is established.

OUT OF PAPER

The *Circuit Cellar* editorial staff has one of those long hooks they used to use to pull performers off stage. That means I'm out of paper, but I'm not done with dialup and PPP. I'll cover the next two phases, authenticate and network, next month. Until then, remember, it doesn't have to be complicated to be embedded. ■

Author's Note: I would like to thank Tamara Kaestner and Charlie Krauter for sharing their wealth of Rabbit knowledge.

Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.

SOURCE

Rabbit 2000 Core Module
Rabbit Semiconductor
(530) 757-8400
Fax: (530) 757-8402
www.rabbitsemiconductor.com